Git

TECHNISCHE UNIVERSITÄT DARMSTADT

Bachelor-Praktikum SS16

Michael Reif, basierend auf Folien von Johannes Lerch

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Idee:** Neues SCM für den Linux Kernel
- **Entwickelt:** 2005
- **Autor:** Linus Torvalds
- **Lizenz:** GPL
- **Sprache:** C, Perl

git [Br.] [coll.] **Penner** *{m}* [ugs.] [widerlicher Typ]

Linus Torvalds:

> […] a rotten person […] I'm an egoistical bastard, so I name all my projects after myself. First Linux, now Git.

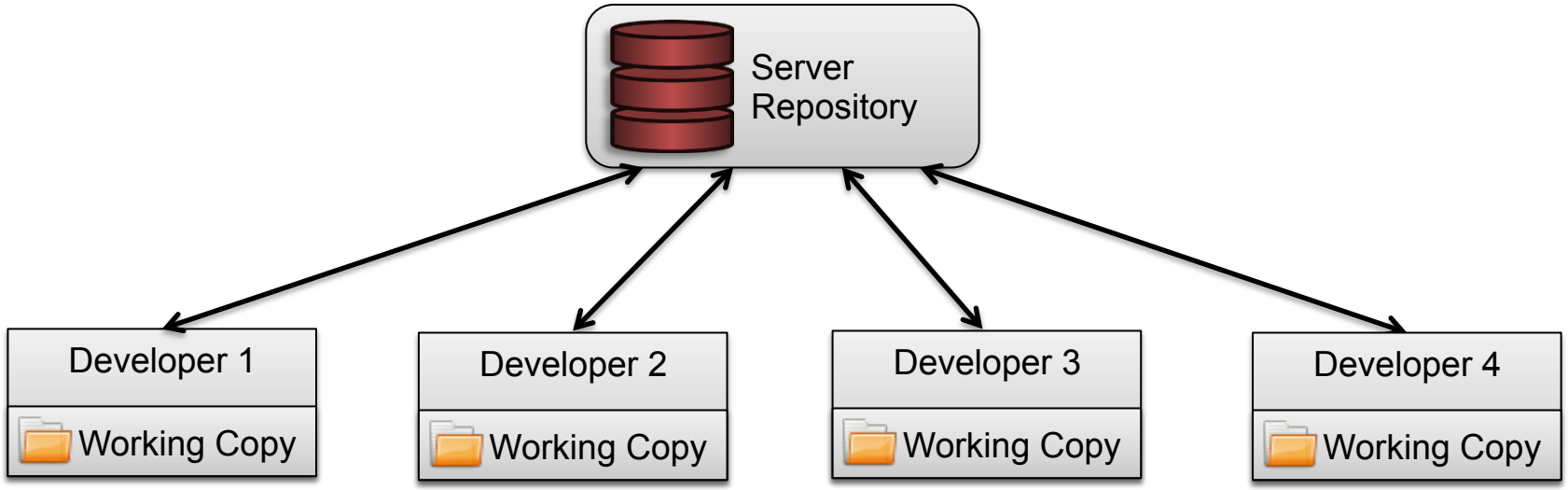
Noch ein SCM?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verteiltes System
  - Schnell, da eine Vielzahl der Operationen lokal ist
  - Ausfallsicher
    - Jeder besitzt alles (= verteiltes Backup)
    - Serverausfall (erstmal) nicht kritisch
- Merge unter Einbezug der Historie
  - Two-Way-Merge (SVN) <--> Three-Way Merge (Git)
- Adaptionsmöglichkeiten an unterschiedliche Workflows

Zentralisiert vs. Verteilt

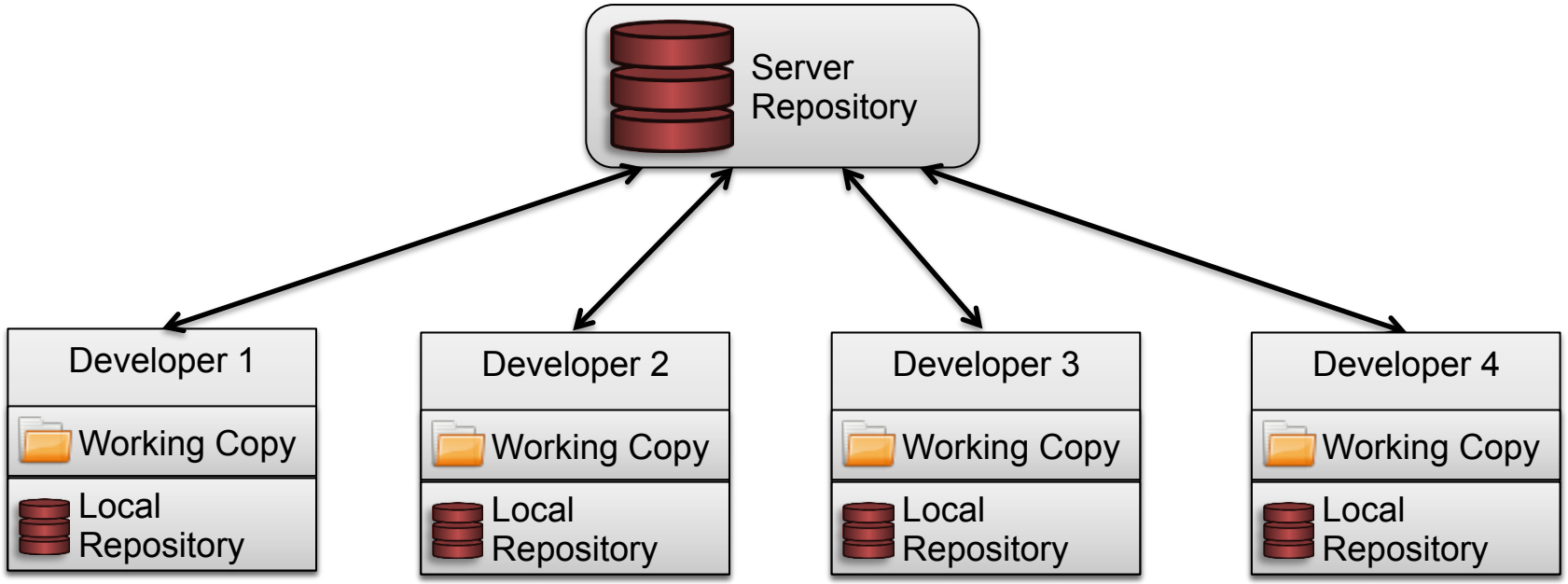
Zentralisiert (SVN)



Zentralisiert vs. Verteilt

Zentralisiert (Git)

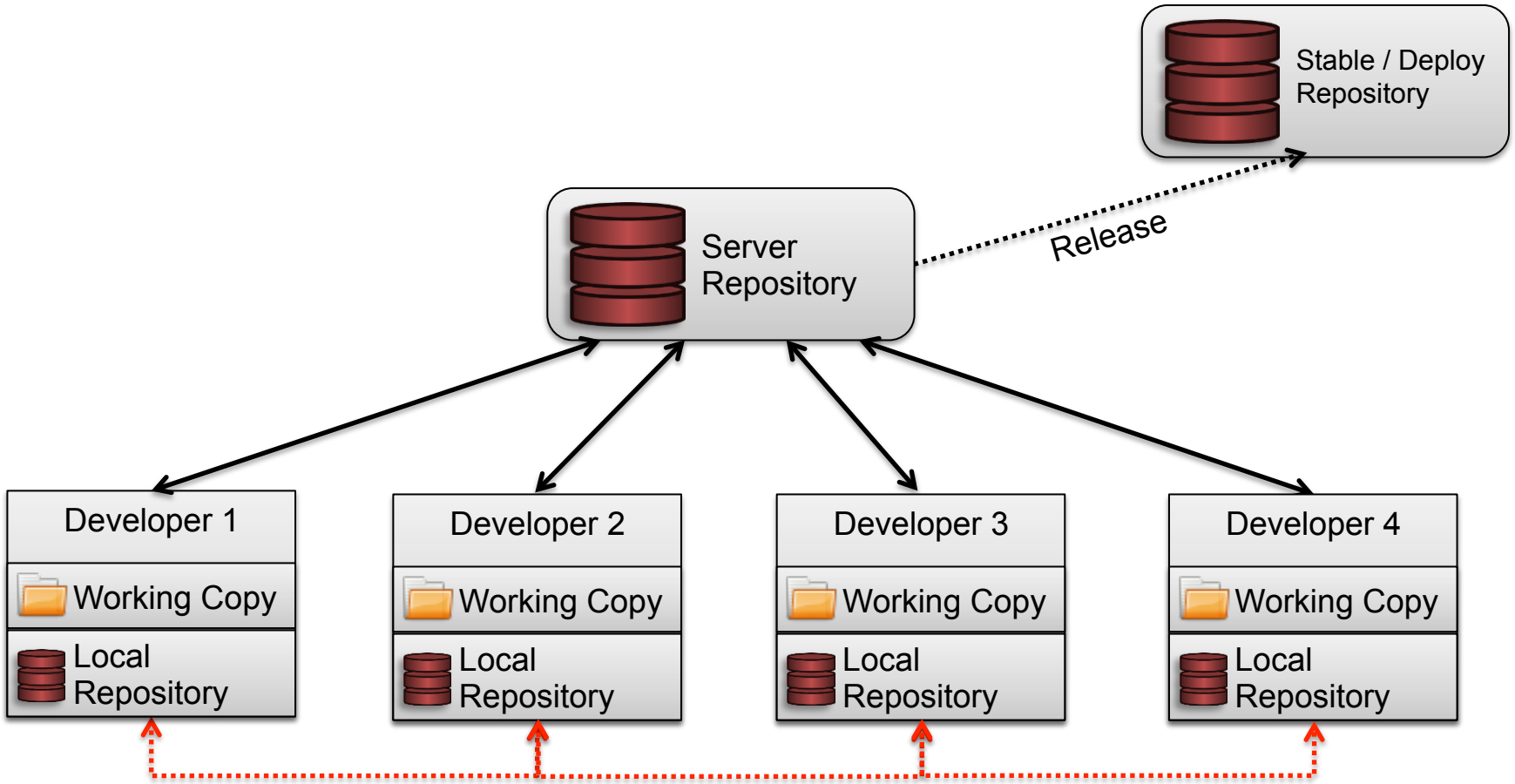
TECHNISCHE UNIVERSITÄT DARMSTADT



Zentralisiert vs. Verteilt

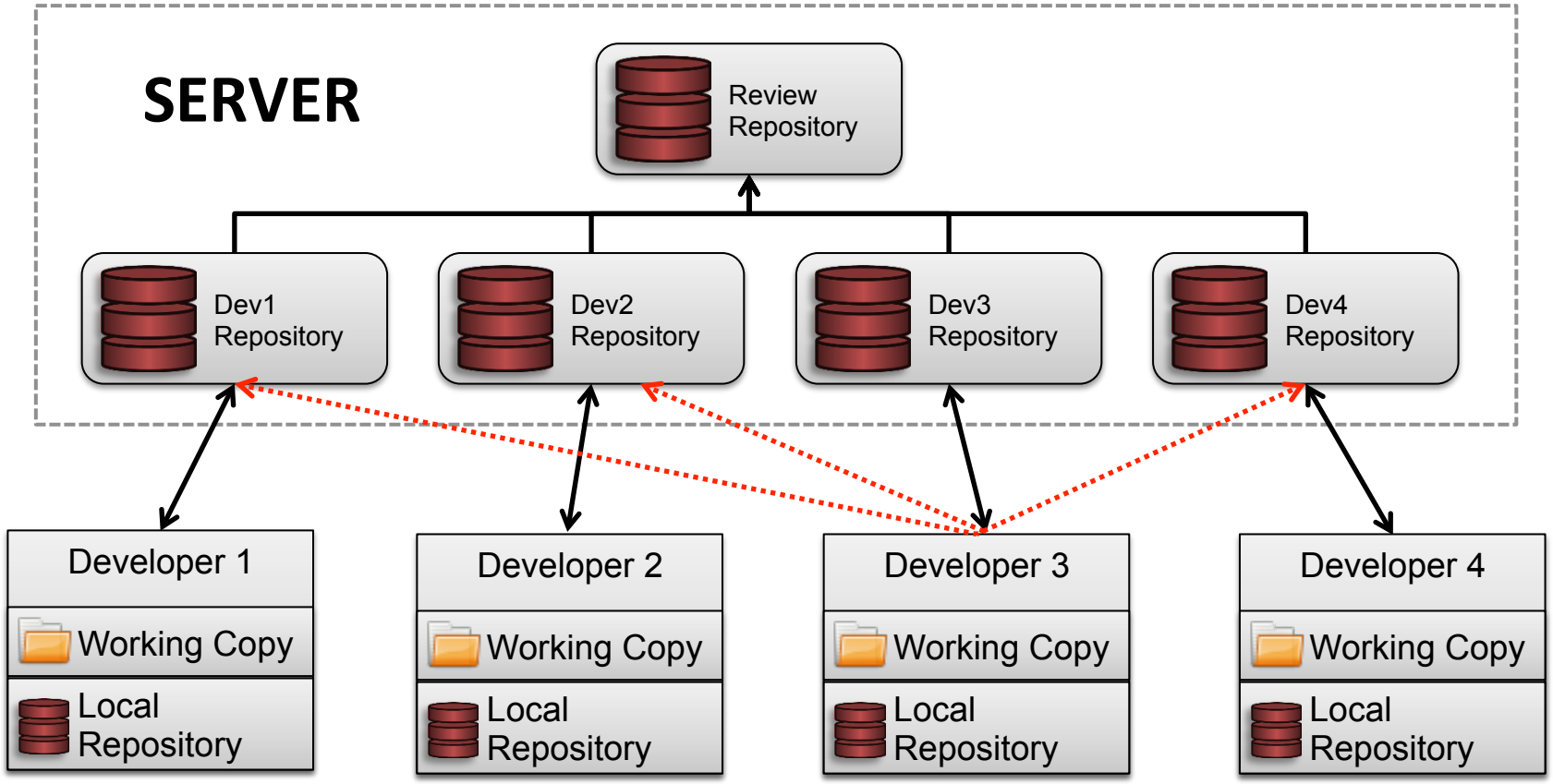
Verteilt (Nur Git)

TECHNISCHE UNIVERSITÄT DARMSTADT

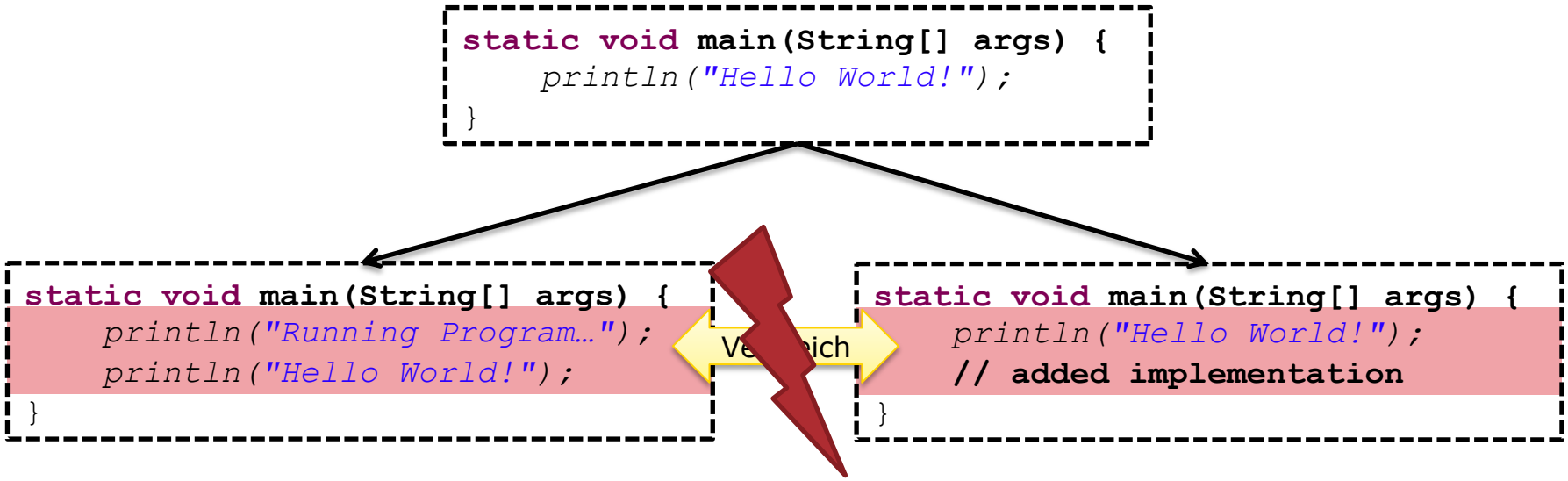


Zentralisiert vs. Verteilt

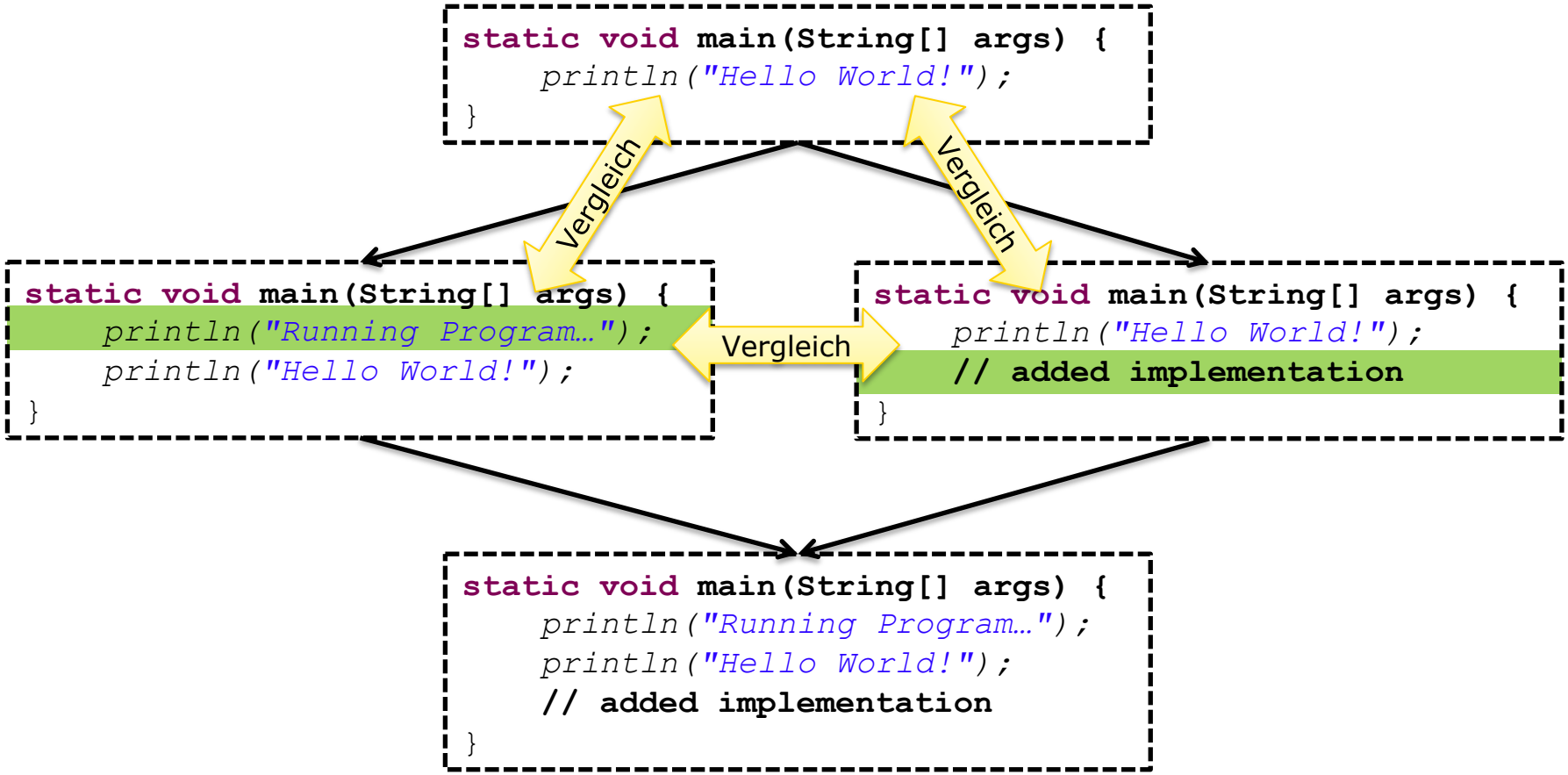
Verteilt² (Nur Git)



Two-Way Merge - Beispiel

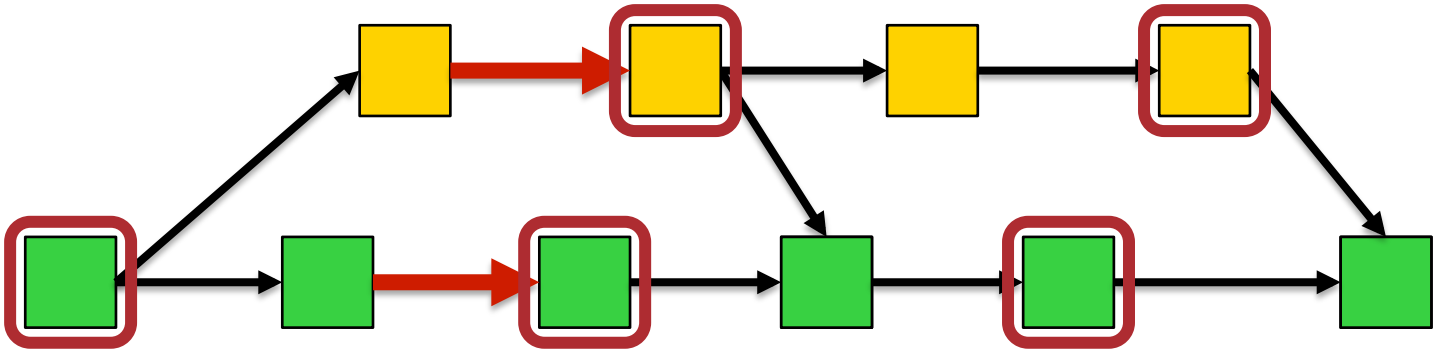


Three-Way Merge - Beispiel

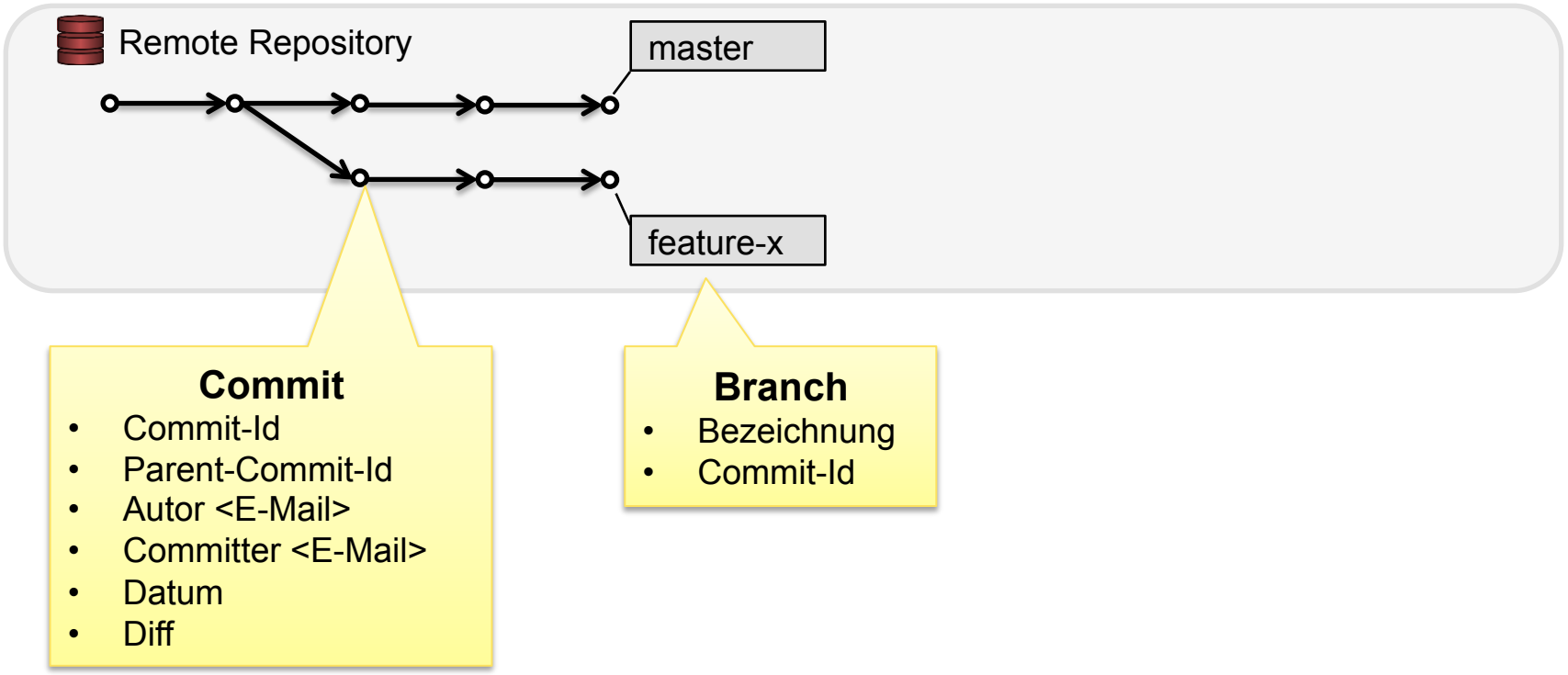


Intelligenter Merge
(Tracking der Versionshistorie)

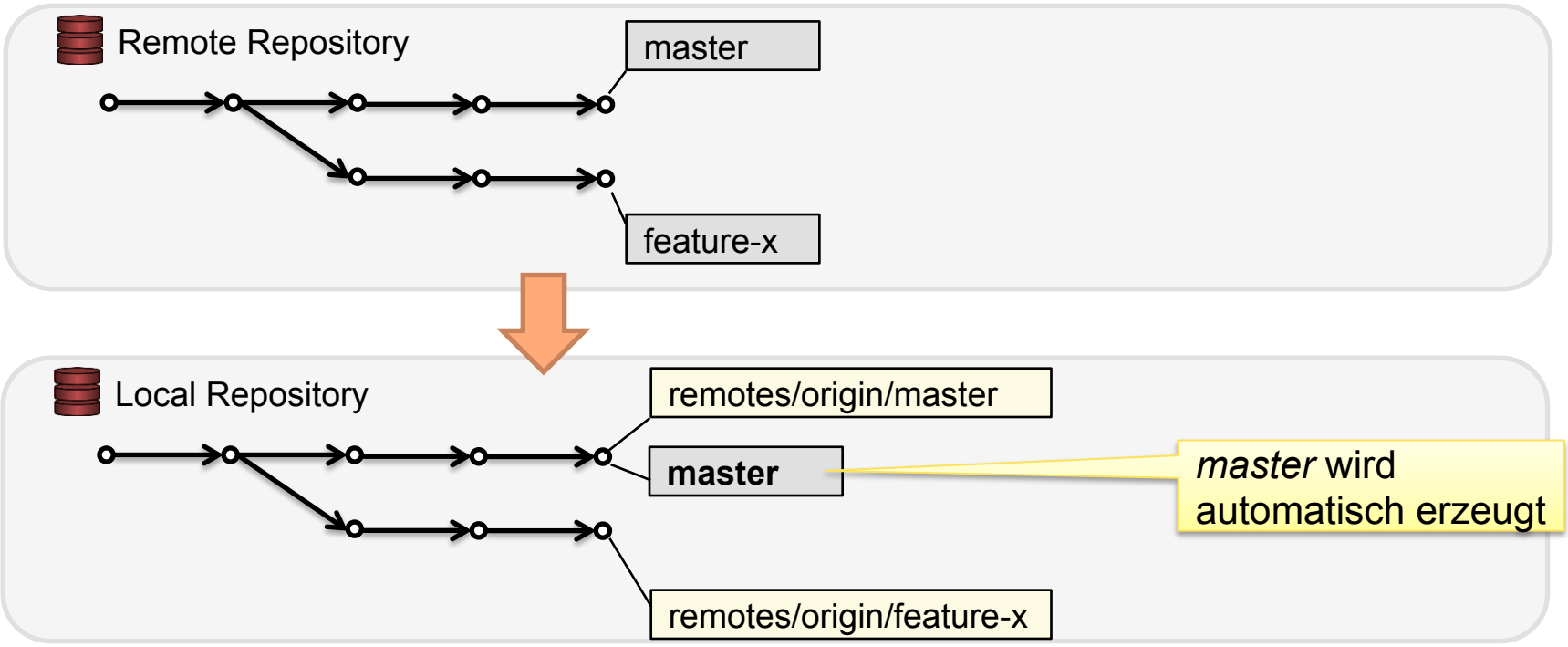
TECHNISCHE UNIVERSITÄT DARMSTADT



Repository

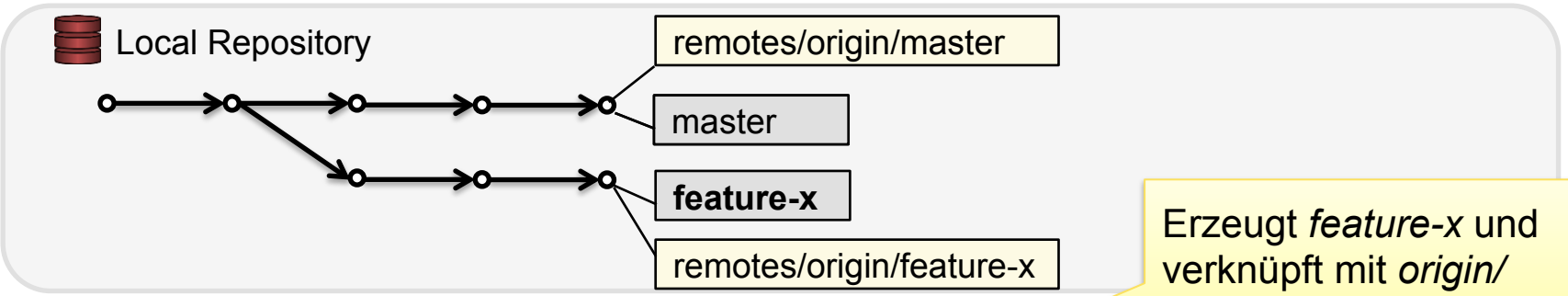

Remote Repository
master
feature-x
Commit
Commit-Id
Parent-Commit-Id
Autor <E-Mail>
Committer <E-Mail>
Datum
Diff
Branch
Bezeichnung
Commit-Id

Clone




```
git clone git://myServer.com/project.git
Cloning into project...
```

Checkout

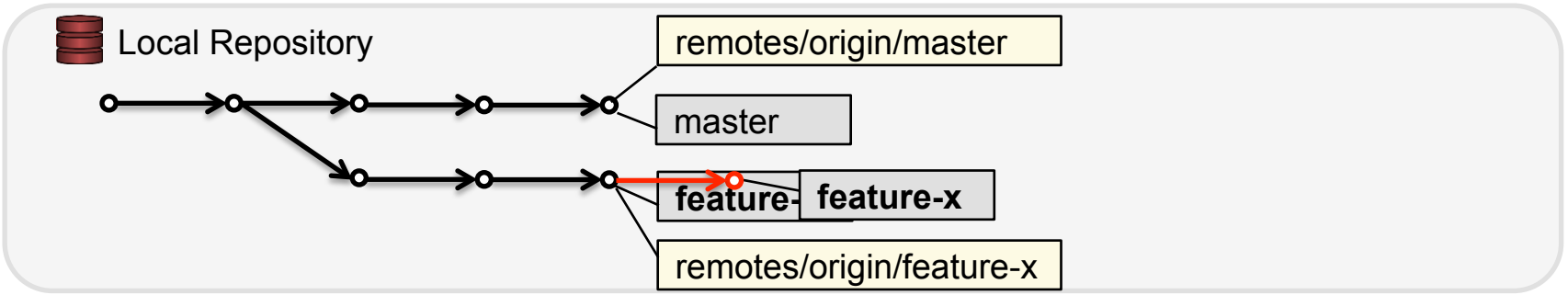



```
git checkout origin/feature-x -b feature-x
Branch feature-x set up to track remote branch
[...]
git checkout feature-x
```

Erzeugt *feature-x* und verknüpft mit *origin/feature-x*

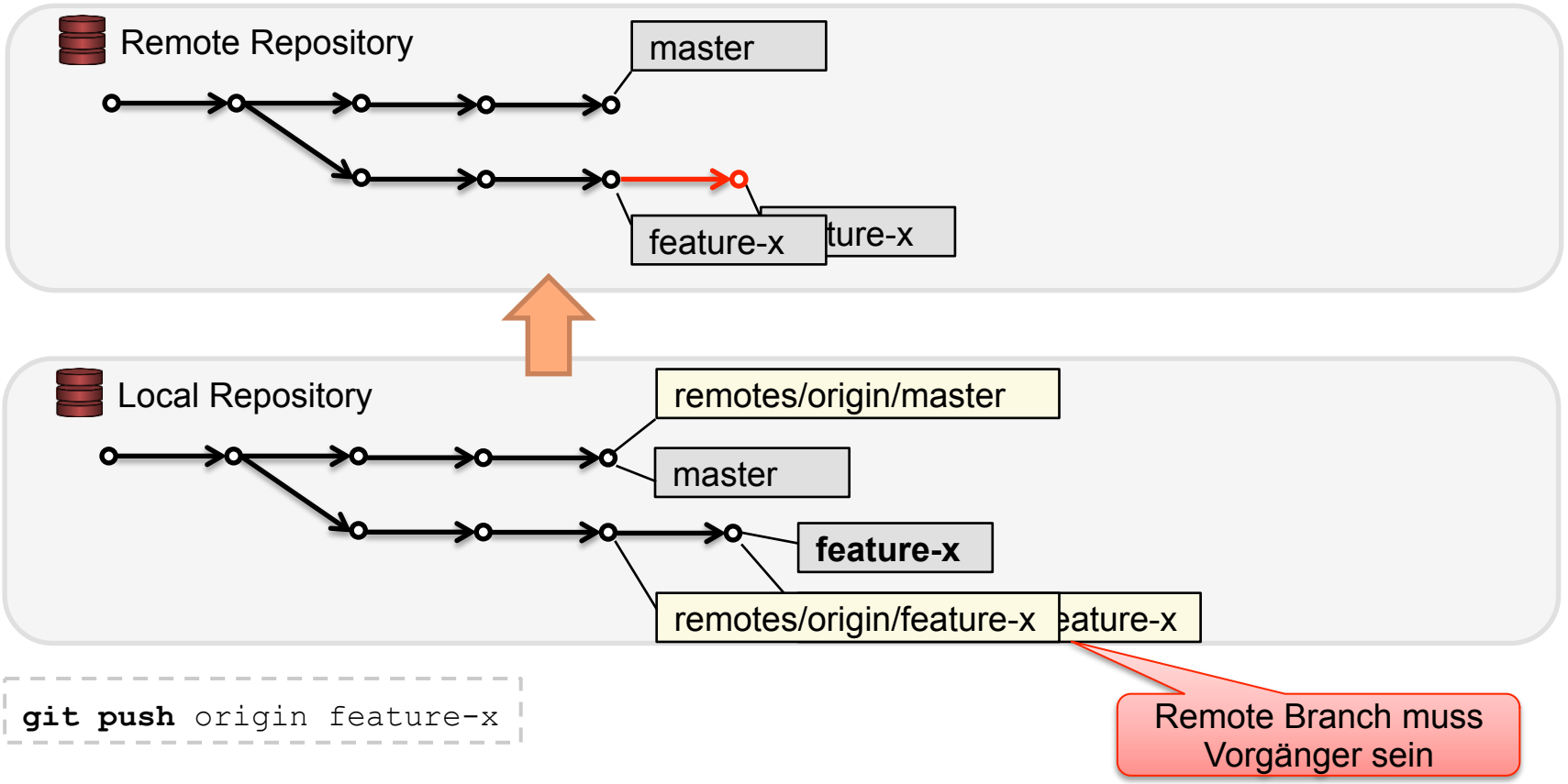
Checkt bestehenden Branch *feature-x* aus

Commit

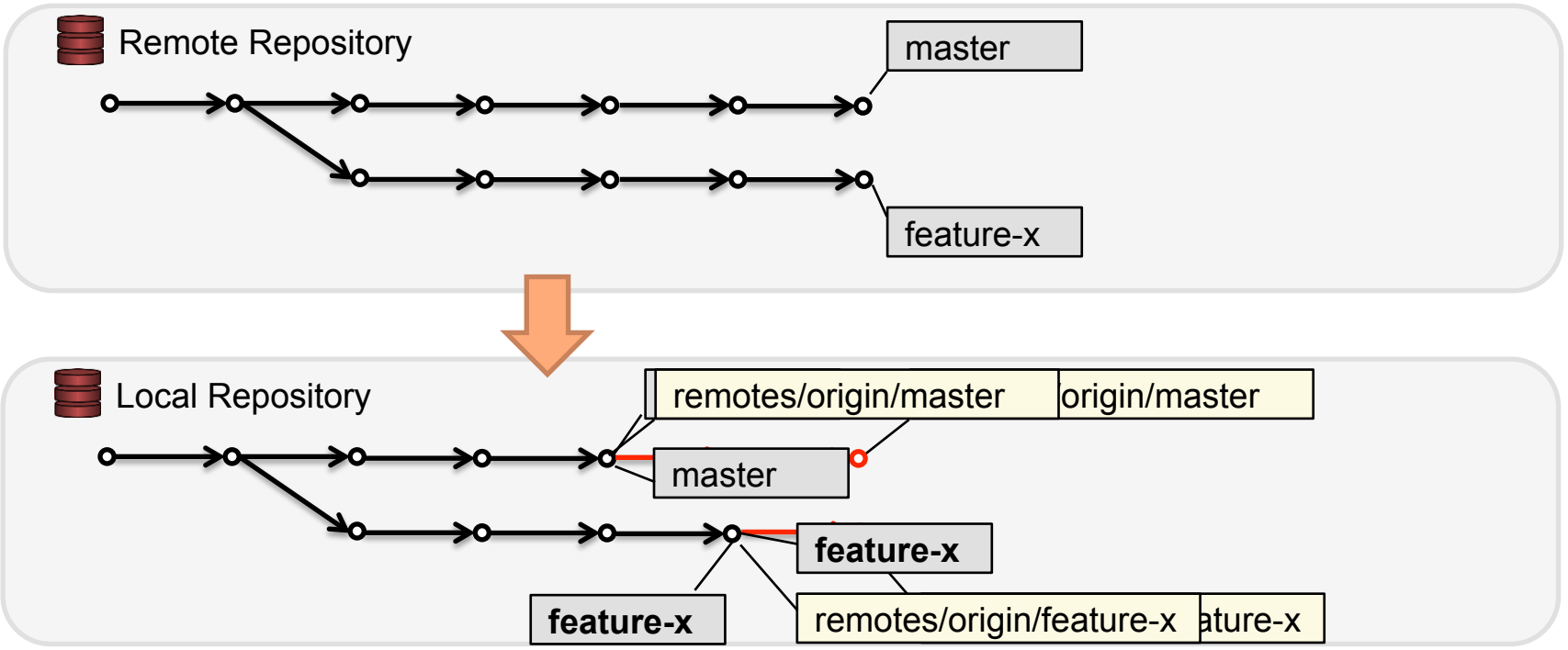



```
touch example.txt
git add example.txt
git commit -m "Commit Message"
```

Push


Remote Repository
master
feature-x
ture-x
Local Repository
remotes/origin/master
master
feature-x
remotes/origin/feature-x
eature-x
git push origin feature-x
Remote Branch muss Vorgänger sein

Fetch



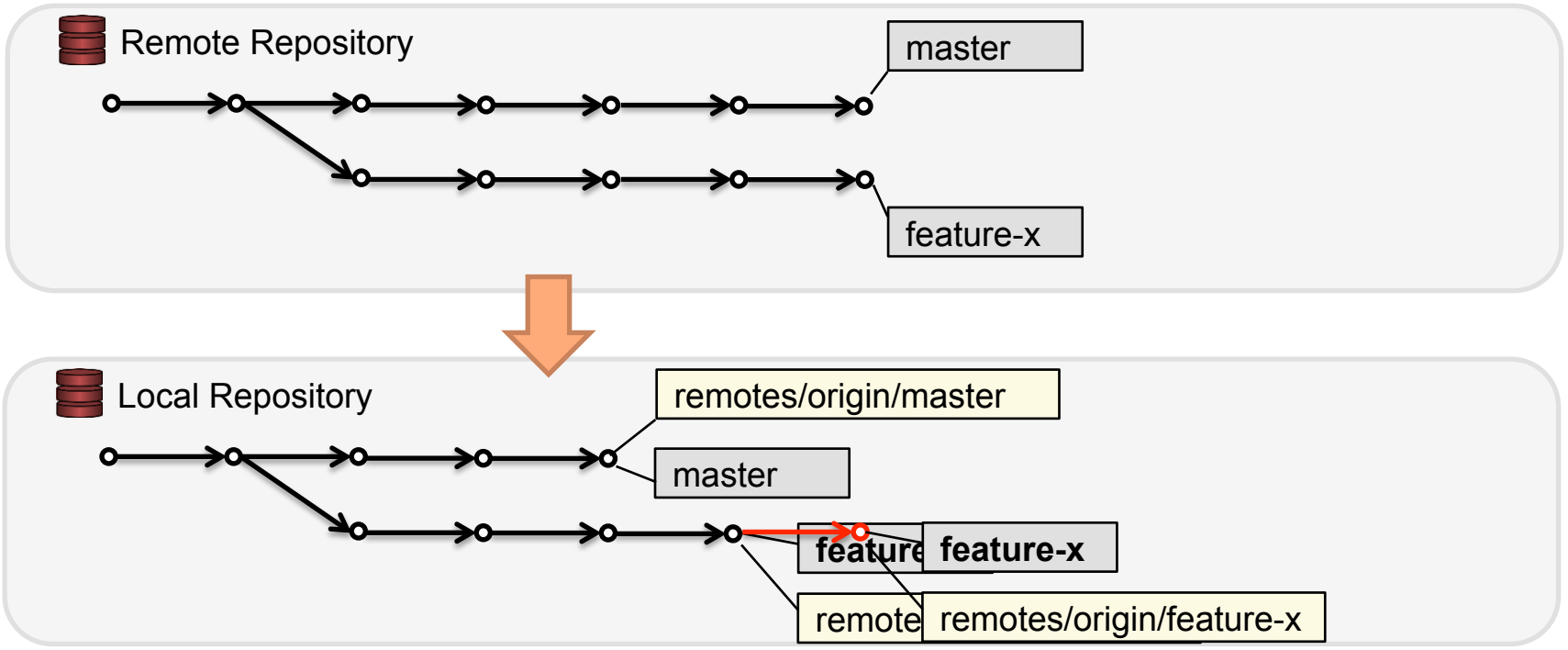

```
git fetch --all
```

Aktualisiert alle Remote Branches im lokalen Repository

```
git fetch
```

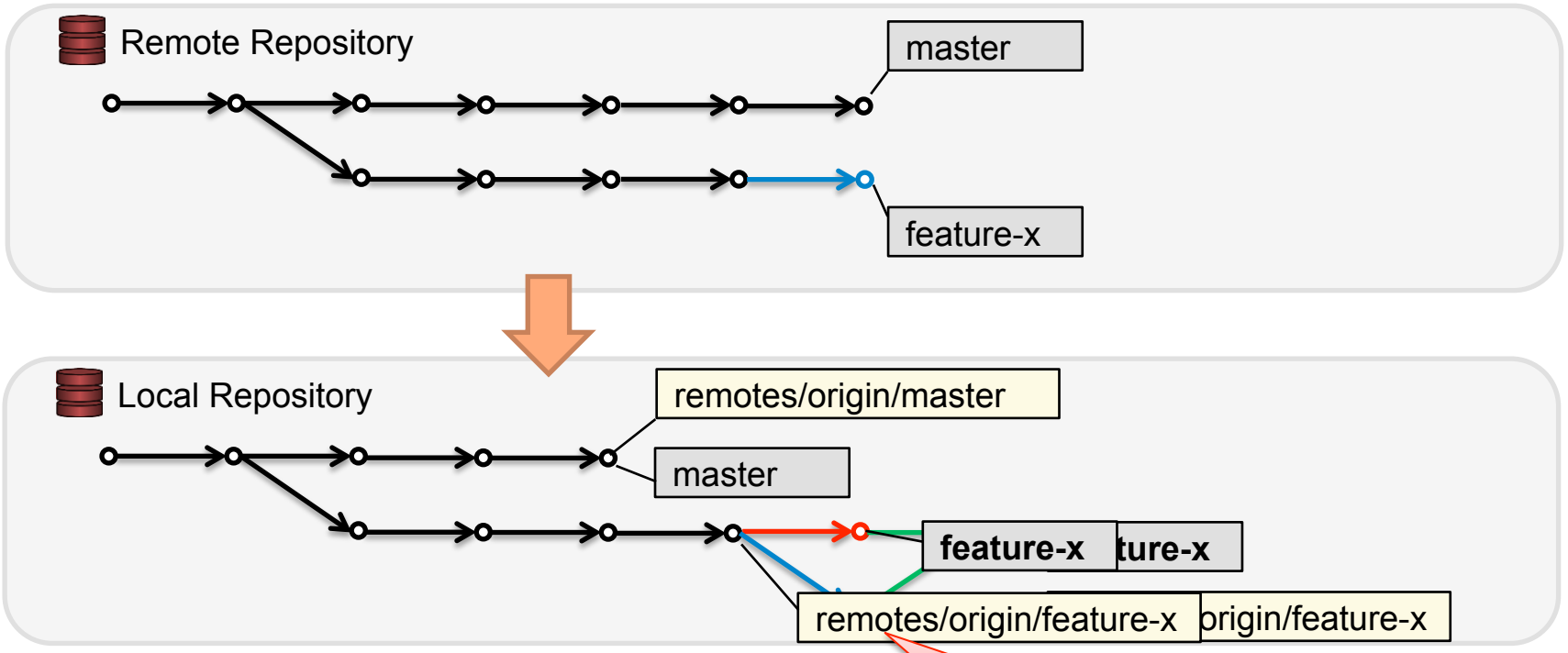
Aktualisiert alle Remote Branches im lokalen Repository des aktuellen Remotes

Pull




```
git pull origin feature-x
```

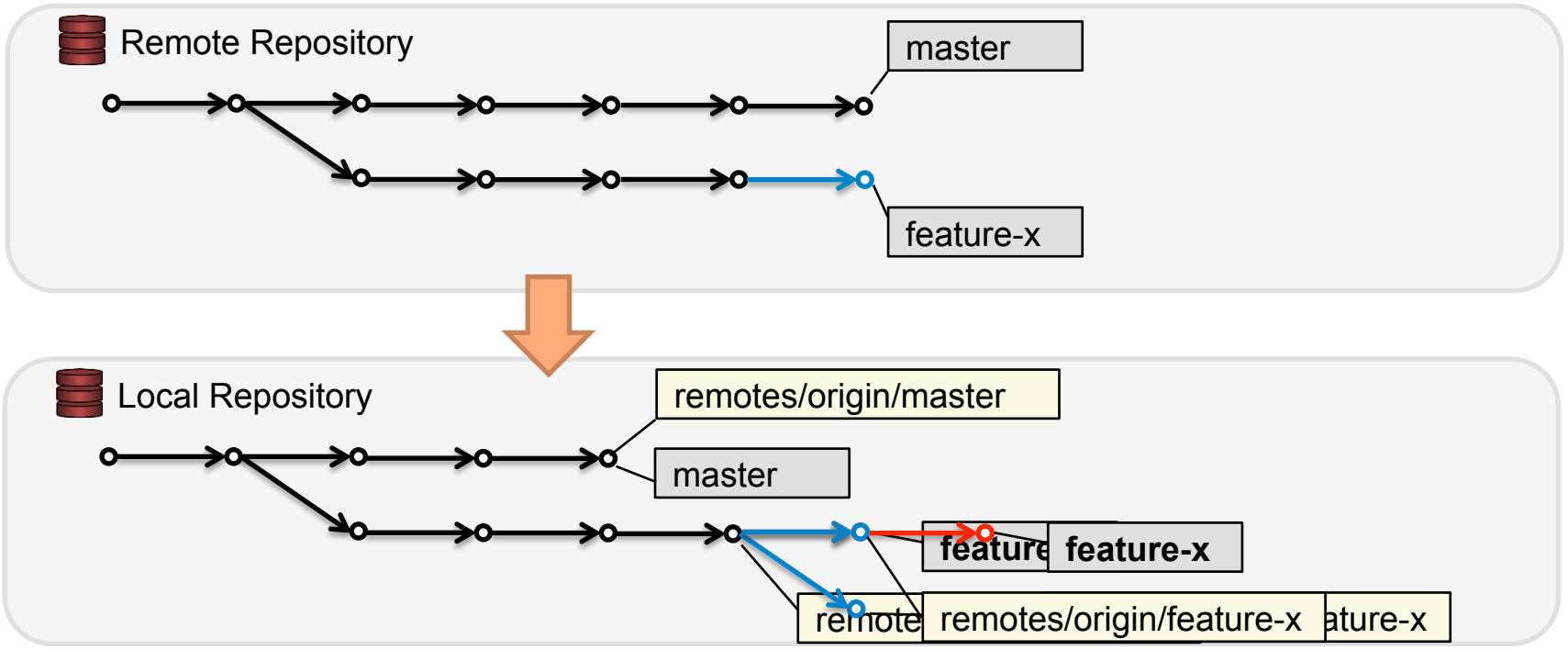
Pull & Merge




```
git pull origin feature-x
CONFLICT (content): Merge conflict in example.txt
Automatic merge failed; fix conflicts and then co
mmit the result.
```

```
nflict
git add example.txt
git commit
```

Pull & Rebase

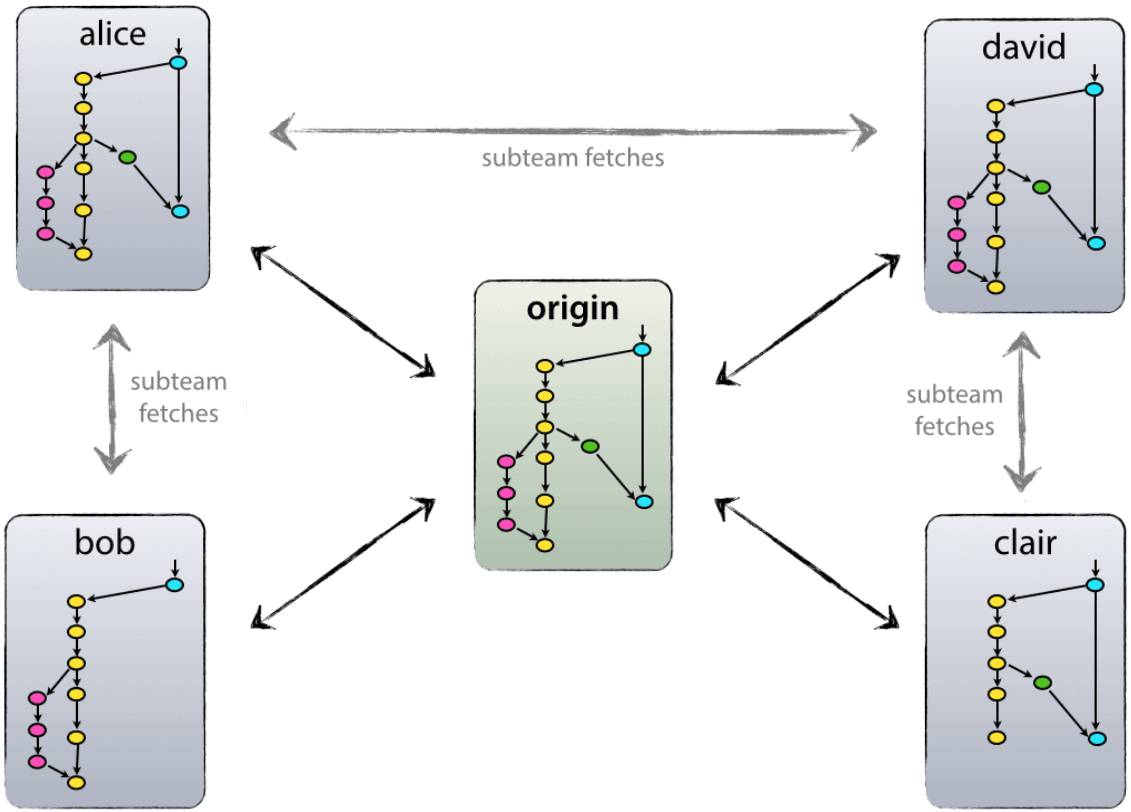



```
git pull --rebase origin feature-x
CONFLICT (content): Merge conflict in example.txt
Failed to merge in the changes.
```

```
// fix conflict
git add example.txt
git rebase --continue
```

Entwicklungsszenario mit GIT

TECHNISCHE UNIVERSITÄT DARMSTADT

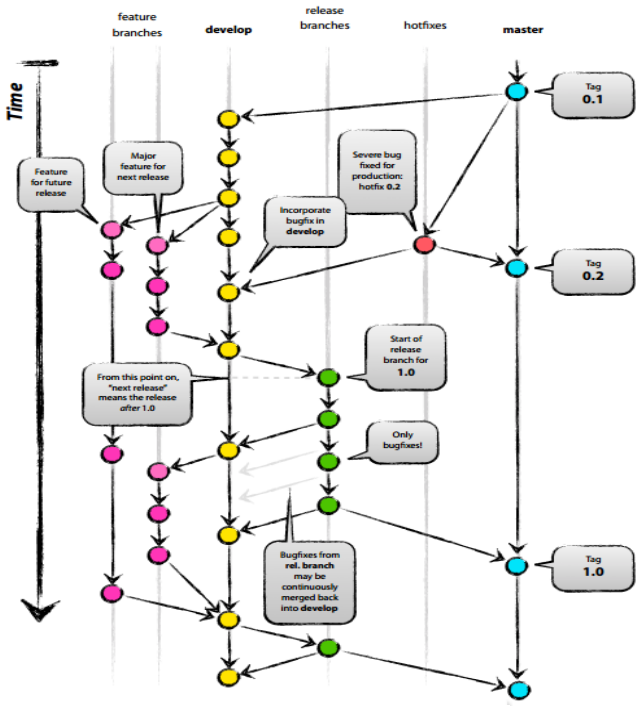


http://nvie.com/posts/a-successful-git-branching-model/

Git-flow als Workflow

TECHNISCHE UNIVERSITÄT DARMSTADT

- Branching Modell von Vincent Driessen
- 2 haupt Branches
  - Branch für produktiv einsetzbare Software
  - Branch für die Entwicklung von Software
- 3 unterstützende Branches
  - Branch für zukünftige oder experimentelle Features
  - Branch für Hotfixes
  - Branch für die Vorbereitung zukünftiger Releases
- TOOLs
  - SourceTree (alle Systeme)
  - Tower (Mac)
  - git-flow (Kommandozeile)

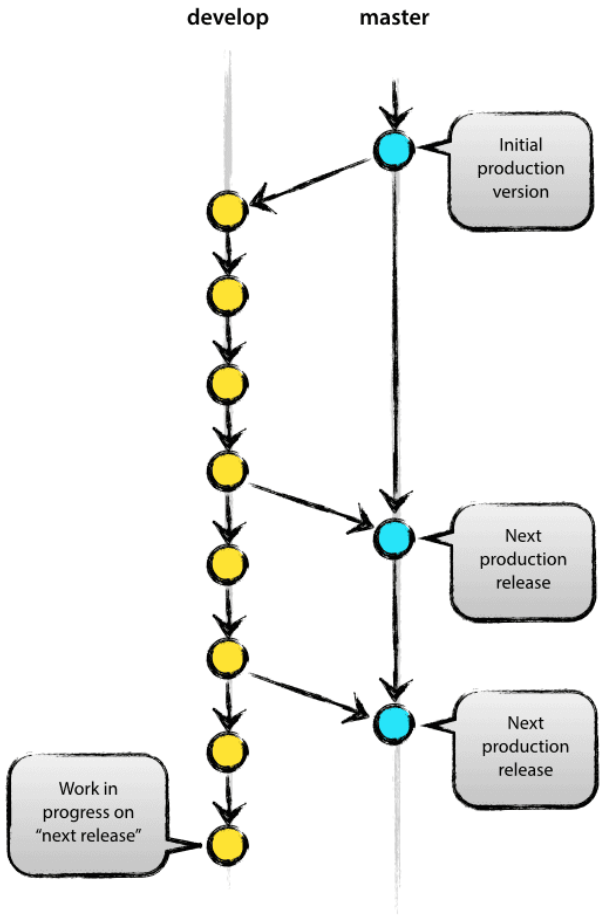


http://nvie.com/posts/a-successful-git-branching-model/

Main Branches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Master-Branch
  - Quellcode entspricht immer einer produktiv einsetzbaren Version
- Develop-Branch
  - Quellcode entspricht der letzten Entwicklungsversion mit neuen aber abgeschlossen Features
  - Testbuilds werden von diesem Branch erstellt
  - Merge in den Master-Branch, wenn ein stabiler Stand erreicht ist



http://nvie.com/posts/a-successful-git-branching-model/

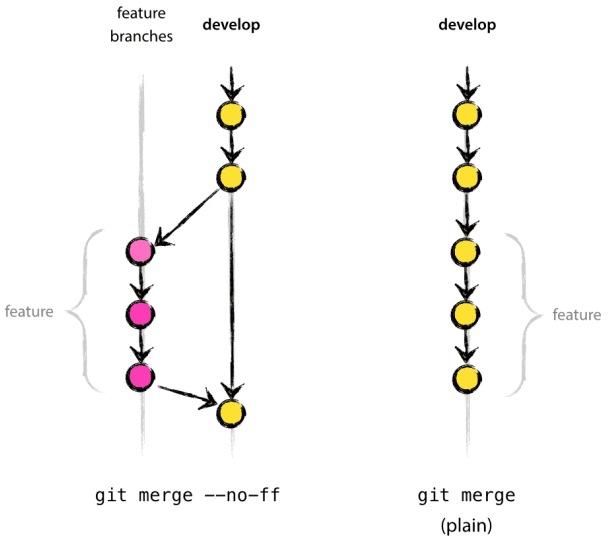
Feature-Branches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Entwicklung von zukünftigen oder experimentellen Features

- Es wird immer von dem develop-Branch erstellt
- Abgeschlossene Features werden zurück gemerged
- Enttäuschende Features werden verworfen

- Namenskonvention: „<Name des Features>“




```
$ git checkout -b myfeature develop
> Switched to a new branch "myfeature"
```

Erstellt einen neuen Branch

http://nvie.com/posts/a-successful-git-branching-model/

Release-Branches

TECHNISCHE UNIVERSITÄT DARMSTADT

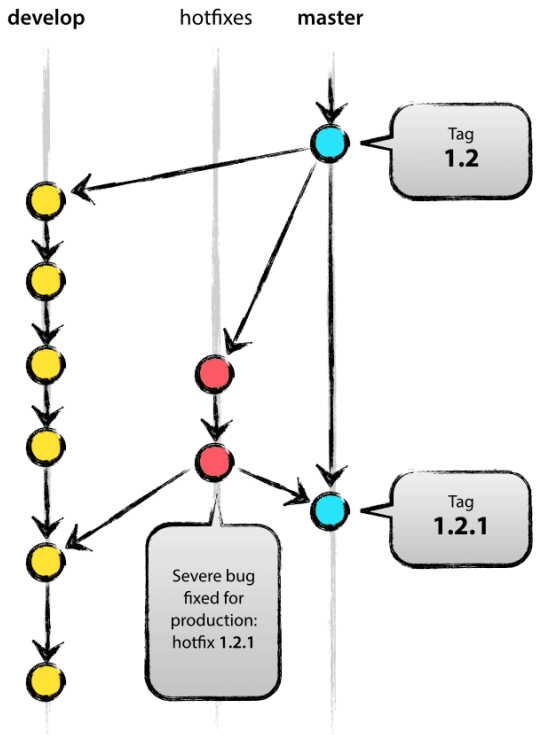
- Unterstützt die Vorbereitung auf neue Releases, wenn der Develop-Branch einen entsprechenden Stand hat
- Ermöglicht kleine Bug fixes für den nächsten Release
- Vorbereitung von Meta-Daten
  - Versionsnummern
  - Konfigurationen
- Wird direkt zurück in den Master- und Develop-Branch gemerged
- Namenskonventions: „release-<versionnumber>“

```
$ git checkout –b release-1.2 develop
> Switched to a new branch “release-1.2“
$ ./bump-version.sh 1.2
> Files modified successfully, version bumped to 1.2.
$ git commit –a –m „Bumped version number to 1.2“
> 1 files changed, 1 insertions(+), 1 deletions(-1)
```

Hotfix-Branches

TECHNISCHE UNIVERSITÄT DARMSTADT

- Tragen wie Release-Branches zu einer neuen Produktionsversion bei
- Kommen zum Einsatz, wenn dringend etwas an der produktiven Software geändert werden muss (z.B. Bugfix)
- Der Branch wird direkt vom Master-Branch erstellt
- Nach Abschluss des Hotfixes wird der Branch sowohl in den Master als auch in den Develop-Branch gemerged
  - Ausnahme: Existiert ein Release-Branch, dann wird der Hotfix in den Master- und Release-Branch gemerged
- Namenskonvention: „hotfix-<versionnumber>“



http://nvie.com/posts/a-successful-git-branching-model/